Short Papers

On the Skew-Bounded Minimum-Buffer Routing Tree Problem

Christoph Albrecht, Andrew B. Kahng, Bao Liu, Ion I. Măndoiu, and Alexander Z. Zelikovsky

***Abstract*—Bounding the load capacitance at gate outputs is a standard element in today's electrical correctness methodologies for high-speed digital very large scale integration design. Bounds on load caps improve coupling-noise immunity, reduce degradation of signal transition edges, and reduce delay uncertainty due to coupling noise (Kahng *et al.* 1998). For clock and test distribution, an additional design requirement is bounding the *buffer skew*, i.e., the difference between the maximum and the minimum number of buffers over all of the source-to-sink paths in the routing tree, since buffer skew is one of the main factors affecting delay skew (Tellez and Sarrafzadeh 1997). In this paper, we consider algorithms for buffering a given tree with the minimum number of buffers under given load cap and buffer skew constraints. We show that the greedy algorithm proposed by Tellez and Sarrafzadeh is suboptimal for nonzero buffer-skew bounds and give examples showing that no bottom-up greedy algorithm can achieve optimality. The main contribution of the paper is an optimal dynamic programming algorithm for the problem. Experiments on test cases extracted from recent industrial designs show that the dynamic programming algorithm has practical running time and saves up to 37.5% of the buffers inserted by Tellez and Sarrafzadeh's algorithm.**

***Index Terms*—Algorithms, buffer insertion, buffer skew, dynamic programming, interconnect design.**

NOMENCLATURE

n	number of sinks, i.e., $n = \|S\|$;
C_w	capacitance of a wire of unit length, which is assumed to be the same for all wires;
C_b	buffer input capacitance, assumed to be the same for all buffers;[1]
C_U	given upper-bound on the capacitive load of each buffer and of the source driver;
c_v	input capacitance of sink or buffer v;
l_e	length of wire segment e;
c_e	capacitance of wire segment e, i.e., $c_e = C_w l_e$;
T_v	subtree of T rooted at v;
$c(T_v)$	lumped capacitance of T_v, i.e., $c(T_v) = \Sigma_{e \in T_v} c_e + \Sigma_{v \in \text{leaves}(T_v)} c_v$;
$l(T_v)$	maximum number of buffers on a path from v to a sink $s \in T_v$ (called longest path in the following);
$s(T_v)$	minimum number of buffers on a path from v to a sink $s \in T_v$ (called shortest path in the following);
$\delta(T_v)$	$l(T_v) - s(T_v)$ (buffer skew of T_v).

Manuscript received September 17, 2001; revised April 13, 2002 and September 27, 2002. This work was supported in part by Cadence Design Systems, Inc., in part by the Microelectronics Advanced Research Corporation (MARCO) Gigascale Silicon Research Center, in part by National Science Foundation Grant CCR-9988331, in part by Moldovan Research and Development Association (MRDA) Award MM2-3018, and in part by the U.S. Civilian Research and Development Foundation for the Independent States of the Former Soviet Union (CRDF), and the State of Georgia's Yamacraw Initiative. This paper was recommended by Associate Editor T. Yoshimura.

C. Albrecht was with the Research Institute for Discrete Mathematics, University of Bonn, 53113 Bonn, Germany. He is now with Synopsys, Inc., Mountain View, CA 94043 USA (e-mail: Christoph.Albrecht@synopsys.com).

A. B. Kahng is with the Department of Computer Science and Engineering and the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@cs.ucsd.edu).

B. Liu was the University of California at San Diego, La Jolla, CA 92093-0114 USA. He is now with Incentia Design Systems, Inc., Santa Clara, CA 95054 USA. (e-mail: bobliu@incentia.com).

I. I. Măndoiu is with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: mandoiu@cs.ucsd.edu).

A. Z. Zelikovsky is with the Computer Science Department, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Digital Object Identifier 10.1109/TCAD.2003.814238

[1]We assume that a single type of buffer is used. Using a single buffer type is a widely accepted design strategy since it reduces process-variation sensitivity and facilitates technology migration.

I. INTRODUCTION

For high-speed digital very large scale integration design, bounding the load capacitance at gate outputs is a standard element in today's *electrical correctness* methodologies. Bounds on load caps improve coupling-noise immunity, reduce degradation of signal transition edges, and reduce delay uncertainty due to coupling noise [6].[2] According to [9], commercial electronic design automation methodologies and tools for signal integrity rely heavily on upper-bounding the capacitive loads on driver and buffer outputs (to prevent very long slew times on signal transitions). Essentially, the load capacitance bounds serve as *proxies* for bounds on input rise/fall times at buffers and sinks (Tellez and Sarrafzadeh [10] formally proves this equivalence using a simple linear model). We assume that such capacitive load bounds are inherent to any buffered routing-tree design task. It is natural to propose a *minimum-buffer* formulation, so as to minimize changes made to the routing tree in meeting the load bounds.

Buffering to control slew times is also critical to *early timing analysis*. With lookup-table-based modeling of gate delays and output transition times, very long input slews tend to be propagated inaccurately, resulting in extremely slow transitions. Static timing analyses that are based on the associated delay calculations will be utterly compromised and useless for driving performance optimizations. Thus, early timing analysis must start with a buffering solution that bounds the capacitive loads of all buffers and the source driver. Again, a *minimum-buffer* objective is appropriate.

Finally, we observe that buffering of some large routing trees (e.g., for clock and test distribution) is further constrained with respect to the *buffer skew*, i.e., the difference between the maximum and the minimum number of buffers over all source-to-sink paths in the routing tree [10]. This is because buffer skew reflects the actual buffered clock-tree skew after routing. To accurately estimate tradeoffs between alternative clock-tree topologies in the early stages of clock distribution design, the key problem is to bound the number of buffers needed by a given tree to satisfy given constraints on both *slew rate* (input rise/fall times) and *buffer skew*. Good bounds (or good constructions that minimize the number of buffers, while controlling the buffer skew) will enable accurate estimation and tradeoff of such system resources as power and area.

[2]Such bounds also improve reliability with respect to hot-carrier oxide breakdown (hot electrons) [4], [5], AC self-heating in interconnects [8], and facilitate technology migration, since designs are more balanced.

From the above context and assumptions, we obtain the following problem formulation.

Bounded Skew Buffering Problem (BSBP): Given a net N, per-unit length wire capacitance, sink and buffer input capacitances, capacitive load bounds for buffers and for the tree source, and an upper bound Δ on buffer skew, find a buffering of N that satisfies all bounds while using the minimum number of buffers.

The BSBP was first formulated by Tellez and Sarrafzadeh [10], who suggested a greedy algorithm with runtime $O(n+k)$, where n is the number of sinks in the net N and k is the number of inserted buffers. In this paper, we make the following contributions.

- We give examples showing the suboptimality of the Tellez-Sarrafzadeh algorithm for BSBP with nonzero buffer-skew bounds, and further, prove that no bottom-up greedy algorithm can achieve optimality (Section III).
- We give a nontrivial dynamic programming algorithm which guarantees optimum solutions for BSBP in $O(n(\Delta+1)^3\mathrm{NB}^2)$ time, where n, Δ, and NB are the number of sinks, the given skew bound, and an upper-bound on the optimum number of inserted buffers, respectively (Section IV).
- We present experimental results on test cases extracted from recent industrial designs, showing that the dynamic programming algorithm has practical running time and inserts up to 37.5% fewer buffers compared to the algorithm in [10] (Section V).

II. Notations and Problem Formulation

We start with a few definitions and notations. Let N be a *net* consisting of a *source* r and a set of *sinks* S.

- A *routing tree* for the net N is a binary[3] tree $T=(r,V,E)$ rooted at r such that each sink of S is a leaf in T.
- A *buffered routing tree* for the net N is a tree $T'=(r,V,E,B)$ such that $T=(r,V,E)$ is a routing tree for N and B is a set of buffers located on the edges[4] of T.
- For any $b\in B\cup\{r\}$, the *subtree driven by* b, D_b, also referred to as the *stage* of b [10], is the maximal subtree of T which is rooted at b and has no internal buffers. A buffered routing tree $T=(r,V,E,B)$ has $|B|+1$ stages including a *source stage* driven by the source.

A. Load Constraints

As noted in [10], bounded slew rate can be ensured by upper-bounding the lumped capacitive load of each buffer and of the source driver. The *lumped capacitive load* of $b\in B\cup\{r\}$ is given by

$$c(D_b)=\sum_{e\in D_b}c_e+\sum_{v\in \mathrm{leaves}(D_b)}c_v. \tag{1}$$

Thus, to ensure bounded slew rate we require that

$$c(D_b)\le C_U \text{ for every } b\in B\cup\{r\}. \tag{2}$$

B. Buffer-Skew Constraints

Tellez and Sarrafzadeh [10] also note that the buffer skew is a significant factor affecting delay skew. Other sources of delay skew, such as propagation delays, have been well studied (heuristics and approximation algorithms for constructing *unbuffered* trees with zero or bounded-skew can be found, e.g., in [3] and [12]). To guarantee bounded delay skew after buffering, we need to ensure that the difference in the number of buffers of the longest and shortest paths from the root r to the sinks is at most a given buffer skew bound Δ, i.e.,

$$\delta(T)=l(T)-s(T)\le\Delta. \tag{3}$$

A buffering satisfying both the load constraint (2) and the buffer skew constraint (3) will be called *feasible*. In this paper, we consider the problem of finding a feasible buffering with minimum number of buffers, formally defined as follows.

C. BSBP

Given: 1) Net N with source r and set of sinks S; 2) binary routing tree $T=(r,V,E)$ for N; 3) sink input capacitances c_s, $s\in S$; 4) buffer input capacitance C_b; 5) unit-length wire capacitance C_w; 6) load upper-bound C_U; and 7) buffer-skew bound Δ.

Find: Buffering $T'=(r,V,E,B)$ of T such that

a) $c(D_b)\le C_U$ for every $b\in B\cup\{r\}$;
b) $\delta(T')\le\Delta$;
c) the total number of inserted buffers $|B|$ is minimum subject to a) and b).

For every $v\in V$, the *branch* of v, denoted $br(v)$, is $T_v\cup(v,\mathrm{parent}(v))$, (where $\mathrm{parent}(r)=r$). If X is a buffering of a subtree containing node v, we denote by X_v the buffering X restricted to the branch $br(v)$.

For each buffering X of a branch $br(v)$, we denote by $nb(X)$, $l(X)$, $s(X)$, and $\mathrm{cap}(X)$ the total number of buffers, the number of buffers on the longest path, the number of buffers on the shortest path, and the residual capacitance (i.e., the capacitance of the stage driven by $\mathrm{parent}(v)$), respectively. Let X and Y be two bufferings of the same branch B. We say that Y *dominates* X if $nb(Y)\le nb(X)$, $l(Y)\le l(X)$, $s(Y)\ge s(X)$, and $\mathrm{cap}(Y)\le cap(X)$. Note that a buffering X of B can be replaced by a buffering Y that dominates it in any context (i.e., in any buffering of the entire routing tree) without increasing the number of buffers or creating load/skew violations.

III. Why Greedy Does Not Work

The BSBP has been previously studied by Tellez and Sarrafzadeh [10]. In [10], a greedy algorithm is first presented for minimum buffering *without* buffer-skew constraints, and then the algorithm is modified to handle such constraints. Below, we describe the two algorithms for the case of binary trees; the description in [10] is given for arbitrary trees.

When there are no constraints on buffer skew, the algorithm in [10] starts with an empty buffering $X=\emptyset$, and then performs the following two steps for each node u, in bottom-up order.

1) *packNode* (u): While $\mathrm{cap}(X_v)+\mathrm{cap}(X_w)>C_U$ (where v and w are the two children of u), add a buffer at the topmost position of the child branch with the largest residual capacitance (the greedy choice).
2) *packEdge* (u): While $\mathrm{cap}(X_u)>C_U$, add a buffer on the edge $(u,\mathrm{parent}(u))$, at the highest possible position still meeting the load cap bound C_U.

With buffer skew constraints, *packEdge* remains the same, while the modified *packNode-BS* (u) consists of the following four steps.

1) Balance T_u as follows. If $l(X_v)<l(X_w)$, then swap v and w. If $l(X_v)-s(X_w)>\Delta$, then insert $l(X_v)-s(X_w)-\Delta$ buffers at the topmost position of $br(w)$. Exit if $\mathrm{cap}(X_u)\le C_U$.

[3]In this paper, we restrict ourselves to binary routing trees Every routing tree can be made binary by duplicating nodes and inserting zero-length edges.

[4]We assume that buffers have a single input and a single output, and thus, are inserted only on the edges of T.

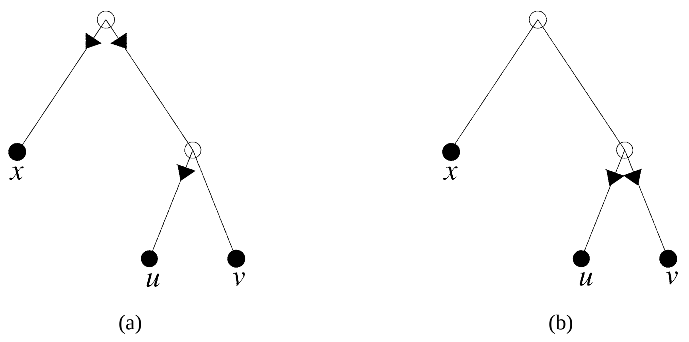

Fig. 1. Counterexample for the greedy BSBP algorithm in [10]. (a) Defined. (b) Defined.

2) Perform *packNode* (u) excluding the child branches with maximum longest path, i.e., if $l(X_w) < l(X_v)$, then add a buffer at the topmost position in $br(w)$. Exit if $\text{cap}(X_u) \leq C_U$.
3) Insert buffers at the topmost position of all child branches with shortest path equal to $l(u) - \Delta$ (in order to maintain buffer skew at most Δ when we insert buffers on the longest paths in the next step). Exit if $\text{cap}(X_u) \leq C_U$.
4) Perform *packNode* (u) considering only child branches with maximum longest path, i.e., longest path equal to $s(u) + \Delta - 1$.

The modified greedy algorithm finds the optimum solution of any given tree when the skew bound Δ is zero. However, contrary to the claim made in [10], the modified greedy algorithm may give suboptimal solutions for $\Delta \geq 1$. There are several reasons for its suboptimality. One reason is that child branches with maximum longest path are considered for buffering *after* considering the other branches, regardless of their residual capacitance. This may cause the algorithm to return a suboptimal solution, e.g., when the skew bound Δ is so large that the buffer-skew constraint never becomes tight (in this case the optimum is found by always choosing the branch with the largest residual capacitance in *packNode*).

Fig. 1 shows a small instance for which the Tellez–Sarrafzadeh algorithm fails to find the optimal buffering. In this instance, we have $\Delta = 1$, $C_w = C_b = 0$, and sink input capacitances are given by $c_u = C_U$ and $c_x = c_v = (3/4)C_U$. Fig. 1(a) shows the suboptimal solution computed by the greedy algorithm, while Fig. 1(b) shows one of the optimal solutions. This instance points to a more basic reason for the suboptimality of the modified greedy algorithm: the optimum buffering of a given tree may be suboptimal when restricted to subtrees.

A natural question prompted by the example in Fig. 1 is whether or not there exists a bottom-up algorithm that computes a *fixed* number of solutions for each branch and still guarantees global optimality. Below, we give two series of examples showing that the answer to this question is negative.

Claim 1: To guarantee optimality, every bottom-up buffering algorithm may need to compute branch bufferings with $m, m+1, \ldots, m+k$ buffers, respectively, where m is the minimum number of buffers for the branch, and k is arbitrarily large.

Claim 1 follows from the example in Fig. 2, in which $\Delta = 1$ and $C_w = C_b = 0$. Each pair of sibling leaves contains a "u" leaf and a "v" leaf, with $c_u = C_U$ and $C_U/(2^{d-2}+1) < c_v < C_U/2^{d-2}$, where d is the depth of T_a.

The minimum number of buffers for each of the two branches into a is 2^{d-2}, since buffers are only required by the "u" leaves. If we start with a minimum number of buffers for both branches into a, we will have to insert a buffer right below a on one of them in order to meet the load constraints. This, in turn, triggers the insertion of a very large number of buffers upstream due to the skew constraint. The optimum overall solution is to insert buffers right above 2^{d-2} of the "v" leaves. This leads to buffering one of the branches into a with at least $(3/2)2^{d-2}$ buffers.

Claim 2: To guarantee optimality, every bottom-up buffering algorithm may need to compute branch bufferings with a longest path equal to $l, l+1, \ldots, l+\Delta-1$, respectively, where l is the minimum longest path.

Claim 2 follows from the example in Fig. 3, in which there are $n = \Delta$ "u" leaves, each with input capacitance $c_u = C_U - 2\varepsilon$, and one additional "v" leaf, with input capacitance $c_v = 0$. We further assume that $C_b = 0$ and that the capacitance of every wire segment in the figure is equal to ε. The n bufferings shown in Fig. 3 have residual capacitance equal to $0, \varepsilon, \ldots, (n-1)\varepsilon$, and a longest path length equal to $n, n-1, \ldots, 1$, respectively. None of these solutions can be dropped from consideration by an optimum algorithm since each of the n different tradeoffs between the longest path length and residual capacitance may be needed upstream.

Indeed, let B_k be the kth buffering (counting from the top) in Fig. 3. B_k has residual capacitance equal to $(k-1)\varepsilon$ and the length of the longest path equal to $n-k+1$. Suppose the upstream topology consists of an edge with total capacitance $k(C_U - \varepsilon)$, connecting a to the source s, and an edge with total capacitance ε connecting to s a sink b with input capacitance $c_b = 0$. If B_k is used to buffer the subtree rooted at a, then a feasible buffering is obtained by inserting $k-1$ buffers between s and a. On the other hand, the following applies.

- If the subtree rooted at a is buffered using B_i, $i > k$, we will need one additional buffer in order to compensate for the larger residual capacitance of B_i.
- If the subtree rooted at a is buffered using B_i, $i < k$, we still need all $k-1$ buffers between s and a to satisfy load-cap constraints. This gives a longest path of $(n-i+1)+(k-1) > n$, and thus, $k-i$ more buffers should be inserted on the edge (s, b) in order to satisfy the buffer-skew constraint.

Thus, B_k is the only buffering from the list in Fig. 3 which can be extended to an optimum buffering under the above upstream topology.

IV. Dynamic Programming Algorithm

In this section, we use dynamic programming to solve the bounded skew-buffering problem. The dynamic programming technique has been applied in the past to timing-driven buffer insertion (see e.g., [1], [7], and [11]). Its application to BSBP presents nontrivial challenges due to the specific buffer-skew constraint. In this section, we first give an exponential time-dynamic programming, then refine it to achieve polynomial time.

A. Exponential Time-Dynamic Programming

The basic observation enabling dynamic programming is that it suffices to consider *normalized* bufferings, i.e., bufferings in which no buffer can be moved higher (closer to the source) on the tree edge to which it belongs. Let NB be the number of buffers inserted in the input tree by the algorithm of Tellez and Sarrafzadeh [10] with the skew-bound set to zero. Clearly, NB is an upper-bound on the number of buffers in any optimal buffering with buffer skew $\Delta > 0$. Thus, we can always guarantee an optimum buffering if we choose the best among the normalized bufferings with up to NB buffers. The exponential time-dynamic programming algorithm, referred to as DP1, computes for each branch $br(u)$, in bottom-up order, the set $\mathcal{L}(u)$ of all normalized bufferings with up to NB buffers.

For a sink u, $\mathcal{L}(u)$ consists of the normalized buffering Z of $br(u) = (u, \text{parent}(u))$ with minimum feasible number of buffers k, plus all bufferings obtained from Z by adding just below parent(u) between 1 and $\text{NB} - k$ buffers, respectively. For a node

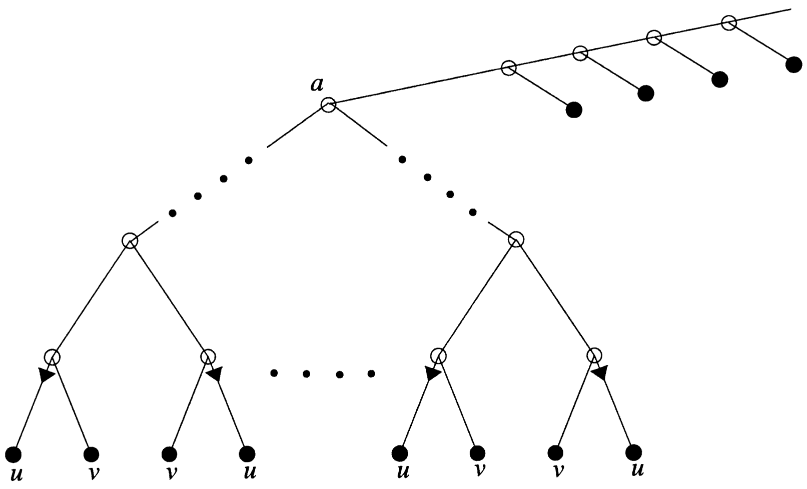

Fig. 2. Proof of Claim 1.

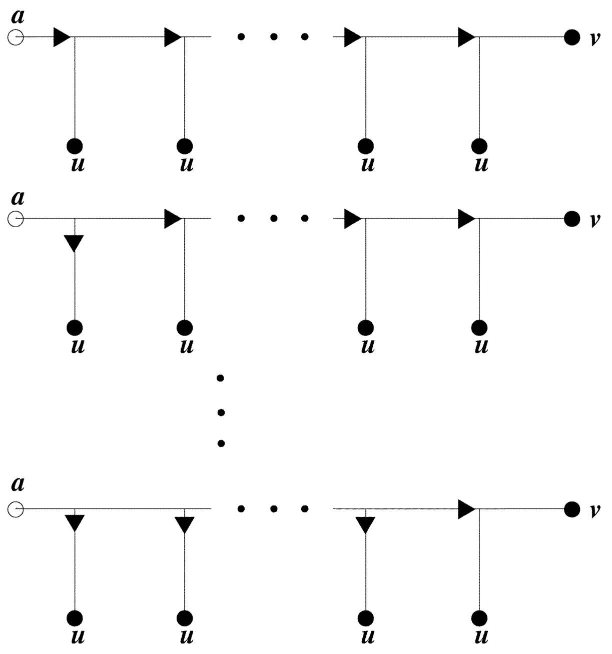

Fig. 3. Proof of Claim 2.

u with children v and w, each buffering of $\mathcal{L}(u)$ is the union of a buffering $X \in \mathcal{L}(v)$, a buffering $Y \in \mathcal{L}(w)$, and a buffering of the edge $(u, \text{parent}(u))$. Each pair of bufferings $(X,\,Y)$ is combined with the buffering of $(u, parent(u))$ with a minimum feasible number of buffers k, as well as all bufferings having between 1 and $\text{NB} - k - nb(X) - nb(Y)$ extra buffers inserted just below $parent(u)$. A pair of bufferings (X, Y) is dropped from consideration if

a) $\text{cap}(X) + \text{cap}(Y) > C_U$ (load cap violation);
b) $nb(X) + nb(Y) + k > \text{NB}$ (too many buffers); or
c) $\max\{l(X) - s(Y), l(Y) - s(X)\} > \Delta$ (skew bound violation).

It is easy to prove by induction that $\mathcal{L}(u)$ contains all normalized bufferings of $br(u)$ with up to NB buffers. Thus, by returning a buffering with a minimum number of buffers from $\mathcal{L}(r)$, DP1 guarantees optimality. The main drawback of DP1 is that, in the worst case, the size of $\mathcal{L}(u)$'s, and hence, the runtime, grows exponentially.[5]

[5]An upper-bound on the size of $\mathcal{L}(u)$ is $\Sigma_{n=0}^{\text{NB}} k^n = O(k^{\text{NB}})$, where k denotes the number of edges in $br(u)$.

B. Polynomial Time-Dynamic Programming

In this section, we describe a polynomial time refinement of DP1, referred to as DP2. In contrast to DP1, DP2 (see Fig. 4) does not add to $\mathcal{L}(u)$ bufferings of $br(u)$ with more than one buffer right below parent(u). More precisely, for each branch $br(u)$, DP2 adds to $\mathcal{L}(u)$ only *nonredundant* bufferings, where a buffering Y of $br(u)$ is said to be *redundant* if there exists a normalized buffering X such that $\text{cap}(X) \le \text{cap}(Y)$, $nb(X) = nb(Y) - k$, $l(X) = l(Y) - k$ and $s(X) = s(Y) - k$, where $k \ge 1$.

For a sink u, $\mathcal{L}(u)$ consists of all nonredundant bufferings of $br(u) = (u, \text{parent}(u))$. There are at most two such nonredundant bufferings: the buffering Z of $(u, \text{parent}(u))$ with a minimum feasible number of buffers, and, if $\text{cap}(Z) > C_b$, the buffering Z' obtained from Z by adding one buffer just below parent(u). Note that the buffering Z' is redundant when $\text{cap}(Z) \le C_b$, since then $\text{cap}(Z) \le \text{cap}(Z') = C_b$, $nb(Z) = nb(Z') - 1$, $l(Z) = l(Z') - 1$, and $s(Z) = s(Z') - 1$.

For a node u with children v and w, let X and Y be bufferings in $\mathcal{L}(v)$, respectively, $\mathcal{L}(w)$. Since redundant bufferings are not explicitly kept as in DP1, DP2 may insert extra buffers at the top of either $br(u)$ or $br(w)$ when combining X and Y. Just as DP1, DP2 drops the pair (X, Y) from consideration when $\text{cap}(X) + \text{cap}(Y) > C_U$ or when combining the pair (X, Y) with the minimum feasible buffering of the edge $(u, \text{parent}(u))$ results in more than NB buffers. If the skew bound for $X \cup Y$ is violated, instead of dropping the pair (X, Y), DP2 fixes the skew by inserting enough buffers at the top of the branch with the fewest buffers on the shortest path. For example, when $l(X) - s(Y) > \Delta$ [see Fig. 5(a)], DP2 inserts $l(X) - s(Y) - \Delta$ buffers at the top of $br(w)$ [see Fig. 5(b)]. Furthermore, DP2 generates more bufferings by inserting extra buffers at the top of the branch with fewest buffers on the shortest path while neither the interval $[s(X), l(X)]$ nor the interval $[s(Y), l(Y)]$ is fully inside the other.[6] For example, for a pair (X, Y) as in Fig. 5(b), extra buffers are inserted one by one at the top of $br(w)$ until either the shortest or longest paths on $br(v)$ and $br(w)$ become equal [see Fig. 5(c)]. Each of these pairs of augmented bufferings of $br(v)$ and $br(w)$ is completed to (at most) two nonredundant bufferings of $br(u)$ by inserting on the edge $(u, \text{parent}(u))$, the minimum feasible number of buffers, and (possibly) one extra buffer just below parent(u).

[6]The bufferings created in this way may be useful since they have smaller skew than $X \cup Y$. On the other hand, the bufferings obtained by continuing to insert buffers after one of the intervals $[s(X), l(X)]$ and $[s(Y), l(Y)]$ encloses the other are dominated by bufferings with these buffers inserted at the top of $(u, \text{parent}(u))$.

Input: Net N with source r and set of sinks S, binary routing tree $T = (r, V, E)$ for N, input capacitances c_s, $s \in S$, buffer input capacitance C_b, unit-length wire capacitance C_w, load upper-bound C_U, buffer-skew bound Δ, and upper bound NB on the number of buffers in an optimal solution
Output: Minimum size feasible buffering of T

```
1. For each $u \in V$, $\mathcal{L}(u) \leftarrow \emptyset$
2. For each sink $s \in S$ do
     Add to $\mathcal{L}(s)$ the buffering $Z$ of $(s, parent(s))$ with minimum feasible number of buffers
     If $cap(Z) > C_b$, add to $\mathcal{L}(s)$ the buffering $Z$ with one more buffer added just below $parent(u)$
3. For each non-sink node $u$ with children $v$ and $w$, in bottom-up order (postorder), do
   (a) For each $X \in \mathcal{L}(v)$ and $Y \in \mathcal{L}(w)$ s.t. $cap(X) + cap(Y) \le C_U$ do
         If $l(X) \ge l(Y)$ then
           $t = \max\{0, l(X) - s(Y) - \Delta\}$
           Repeat forever
             Let $W$ be the buffering of $br(u)$ obtained from $X \cup Y$ by
               (*) adding $t$ buffers at the top of $br(w)$, and
               (**) buffering the edge $(u, parent(u))$ with minimum feasible number of buffers
             If $nb(W) \le NB$ then add $W$ to $\mathcal{L}(s)$
             If $cap(W) > C_b$ and $nb(W) + 1 \le NB$ then add to $\mathcal{L}(s)$ the buffering $W$ with one more buffer added at the top of $br(u)$
             If $nb(W) \ge NB$ or one of the intervals $[s(X), l(X)]$ and $[s(Y) + t, l(Y) + t]$ is inside the other, then exit the repeat loop
             Else $t = t + 1$
         If $l(X) < l(Y)$, repeat above code in which $X$ and $br(v)$ reverse roles with $Y$ and $br(w)$
   (b) Remove from $\mathcal{L}(u)$ all dominated bufferings
   (c) For each buffering $W \in \mathcal{L}(u)$ remove from $\mathcal{L}(u)$ all bufferings $Z'$ with $nb(Z') = nb(W) + k$, $l(Z') = l(W) + k$, and $s(Z') = s(W) + k$, where
$k \ge 2$
4. Return the buffering $X \in \mathcal{L}(r)$ with minimum $nb(X)$
```

Fig. 4. DP2 algorithm for BSBP.

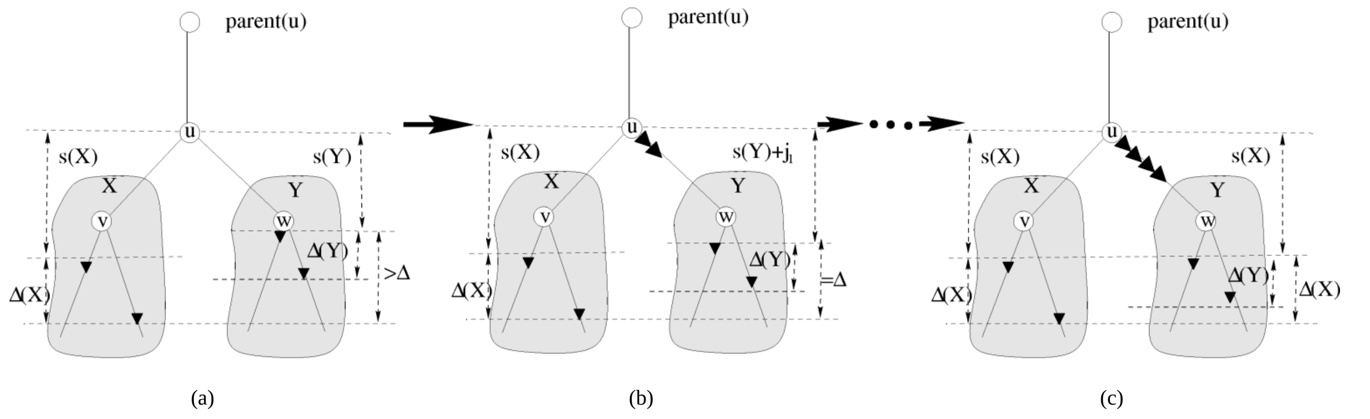

Fig. 5. (a) Pair (X, Y) of bufferings for which the skew is greater than Δ, i.e., $l(X) - s(Y) > \Delta$.(b) The skew is fixed by adding $t = l(X) - s(Y) - \Delta$ buffers at the top of $br(w)$, after this addition $l(X) - s(Y) = \Delta$. (c) More useful skews are generated by adding buffers at the top of $br(w)$. We stop when either shortest or longest paths on $br(v)$ and $br(w)$ become equal (with the former case represented here).

Finally, DP2 refines the set $\mathcal{L}(u)$ by removing all of the dominated [Step 3(b)] and redundant bufferings [Step 3(c)].[7]

Correctness of DP2 follows from the following.

Theorem 1: For each buffering Z of $br(u)$, there exists buffering $Z' \in \mathcal{L}(u)$ and $k \ge 0$ such that Z is dominated by Z' with k buffers added at the top.

Proof: The proof is by induction on the depth of u. The claim is trivially true when u is a sink, i.e., a leaf of T. Assume that the lemma holds for the two children v and w of u. Let X and Y be the restrictions of Z to $br(v)$ and $br(w)$. By induction, there exist $X' \in \mathcal{L}(v)$, $Y' \in \mathcal{L}(w)$, and i, $j \ge 0$, such that X and Y are dominated by X' and Y' with i (respectively, j) buffers added at the top of the respective branches. Additionally, we can assume that i and j are the minimum numbers of redundant buffers with the above property.

Let Z' be the buffering of $br(u)$ obtained from Z by replacing X and Y by X' (respectively, Y') with i (respectively, j) buffers added at the top of $br(v)$ [respectively, $br(w)$]. Clearly, Z' dominates Z. To complete the proof, we need to show that Z' is added by DP2 to $\mathcal{L}(u)$.

[7]This refinement is required since dominated or redundant solutions may be added to $\mathcal{L}(u)$ by combining different pairs (X, Y).

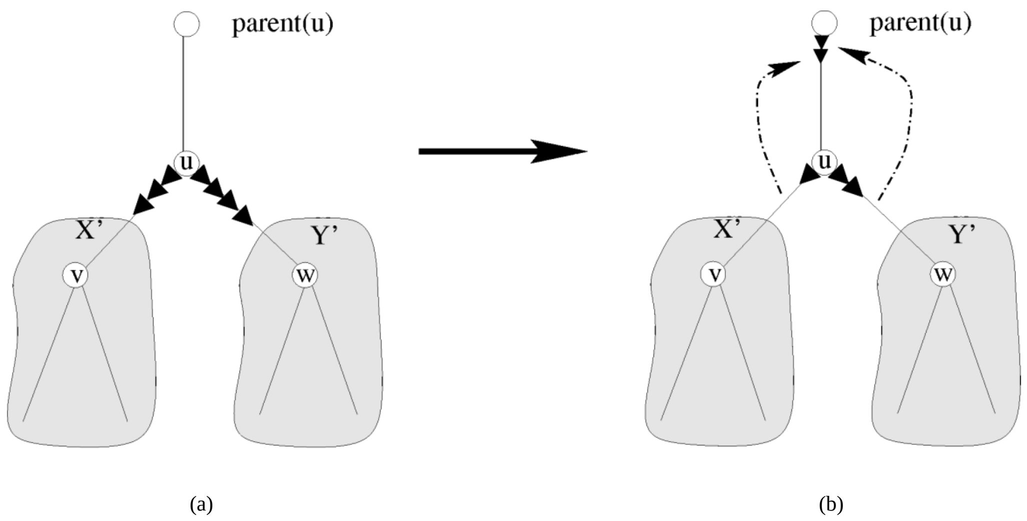

Fig. 6. (a) Redundant buffers on the top of both branches $br(v)$ and $br(w)$. (b) Buffering with redundant buffers moved up to the top of $br(u)$.

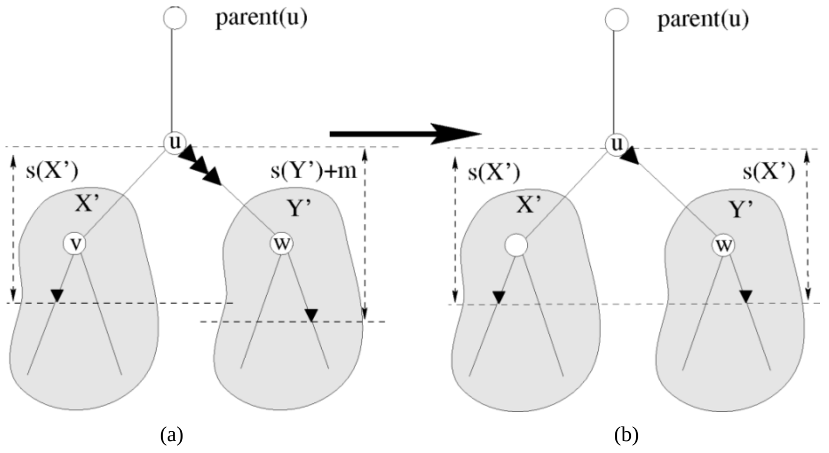

Fig. 7. (a) Pair of bufferings for which $s(X') < s(Y') + j$. (b) After removing excessive redundant buffers we get $s(X') \geq s(Y') + j$.

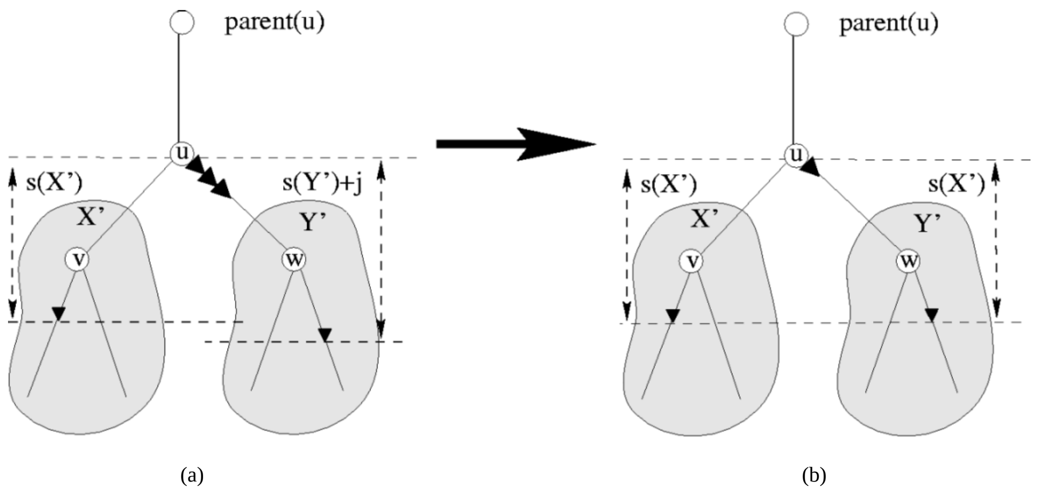

Fig. 8. (a) Pair of bufferings for which $s(X') \geq s(Y') + j$, but $l(X') < l(Y') + j$. (b) Buffering obtained after excessive redundant buffers are moved up from $br(w)$ to the top of $br(u)$.

It is easy to see that this is true when $i = j = 0$. If both i and j are positive, then we can replace Z' with the buffering obtained by deleting $\min\{i, j\}$ redundant buffers from the top of each of the branches $br(v)$ and $br(w)$, and inserting $\min\{i, j\}$ redundant buffer at the top of $br(u)$ (see Fig. 6). Without loss of generality, in the following, we assume that $i = 0$ and $j > 0$.

If $s(X') < s(Y') + j$, then Z' can be replaced by the buffering Z'' obtained by removing $k = \min\{j, s(Y') + j - s(X')\}$ redundant buffers from the top of $br(w)$, which dominates Z. Indeed, $nb(Z'') = nb(Z) - k$, $l(Z'') \leq l(Z)$, and $\text{cap}(Z'') = \text{cap}(Z)$ (because the removed buffers are redundant). Finally, $s(Z'') = s(Z)$, since $s(X') < s(Y') + j$ (see Fig. 7). If $k = j$, then Z'' is added by DP2 to $\mathcal{L}(u)$

TABLE I
NUMBER OF BUFFERS (BOLDFACE) AND RUNTIME (IN SECONDS) FOR DP2 AND THE GREEDY ALGORITHM IN [10] FOR TREES CONSTRUCTED USING GREEDY-DME WITH LINEAR DELAY

Testcase	C_U	$\Delta=0$		$\Delta=1$			$\Delta=2$			$\Delta=3$			$\Delta=4$			Lower
Parameters	(fF)	Greedy	DP2	Greedy	DP2	Gain	Greedy	DP2	Gain	Greedy	DP2	Gain	Greedy	DP2	Gain	Bound
#Sinks=330	500	**34**	**34**	**31**	**26**	**16.1%**	**27**	**25**	**7.4%**	**27**	**25**	**7.4%**	**27**	**25**	**7.4%**	**25**
		0.01	0.02	0.00	0.03		0.01	0.05		0.00	0.07		0.00	0.09		0.00
WL=42mm	1000	**19**	**19**	**17**	**14**	**17.6%**	**15**	**12**	**20.0%**	**16**	**12**	**25.0%**	**16**	**12**	**25.0%**	**12**
		0.01	0.01	0.00	0.03		0.01	0.06		0.00	0.07		0.00	0.09		0.01
C_b=37.50fF	2000	**8**	**8**	**7**	**6**	**14.3%**	**7**	**5**	**28.6%**	**7**	**5**	**28.6%**	**7**	**5**	**28.6%**	**5**
		0.01	0.01	0.00	0.03		0.01	0.05		0.00	0.07		0.00	0.09		0.00
Min/Max/Σc_s(fF)	4000	**5**	**5**	**4**	**3**	**25.0%**	**3**	**3**	**0.0%**	**3**	**3**	**0.0%**	**3**	**3**	**0.0%**	**3**
		0.00	0.01	0.00	0.03		0.01	0.05		0.01	0.07		0.00	0.08		0.00
2.04/27.75/3017	8000	**2**	**2**	**1**	**1**	**0.0%**	**1**	**1**	**0.0%**	**1**	**1**	**0.0%**	**1**	**1**	**0.0%**	**1**
		0.00	0.01	0.00	0.03		0.01	0.04		0.01	0.07		0.00	0.09		0.00
#Sinks=830	500	**101**	**101**	**91**	**87**	**4.4%**	**90**	**83**	**7.8%**	**89**	**81**	**9.0%**	**88**	**81**	**8.0%**	**81**
		0.02	0.03	0.01	0.04		0.01	0.08		0.01	0.10		0.01	0.11		0.01
WL=170mm	1000	**46**	**46**	**45**	**41**	**8.9%**	**45**	**38**	**15.6%**	**43**	**38**	**11.6%**	**42**	**38**	**9.5%**	**38**
		0.01	0.02	0.01	0.04		0.01	0.06		0.01	0.08		0.01	0.10		0.01
C_b=37.50fF	2000	**25**	**25**	**23**	**20**	**13.0%**	**24**	**19**	**20.8%**	**23**	**19**	**17.4%**	**23**	**19**	**17.4%**	**19**
		0.01	0.02	0.01	0.05		0.01	0.07		0.01	0.10		0.01	0.12		0.00
Min/Max/Σc_s(fF)	4000	**11**	**11**	**9**	**9**	**0.0%**	**9**	**9**	**0.0%**	**10**	**9**	**10.0%**	**10**	**9**	**10.0%**	**9**
		0.01	0.02	0.01	0.04		0.01	0.07		0.01	0.09		0.01	0.10		0.01
4.25/4.25/3528	8000	**5**	**5**	**5**	**4**	**20.0%**	**4**	**4**	**0.0%**	**4**	**4**	**0.0%**	**4**	**4**	**0.0%**	**4**
		0.02	0.02	0.02	0.05		0.01	0.06		0.01	0.09		0.01	0.11		0.01
#Sinks=1900	500	**105**	**105**	**90**	**84**	**6.7%**	**87**	**79**	**9.2%**	**87**	**79**	**9.2%**	**85**	**78**	**8.2%**	**78**
		0.03	0.08	0.03	0.14		0.03	0.22		0.03	0.30		0.03	0.37		0.02
WL=76mm	1000	**53**	**53**	**46**	**42**	**8.7%**	**43**	**40**	**7.0%**	**43**	**39**	**9.3%**	**44**	**39**	**11.4%**	**39**
		0.02	0.05	0.02	0.12		0.03	0.21		0.03	0.29		0.02	0.35		0.02
C_b=37.50fF	2000	**26**	**26**	**23**	**20**	**13.0%**	**22**	**19**	**13.6%**	**21**	**19**	**9.5%**	**21**	**19**	**9.5%**	**19**
		0.03	0.05	0.03	0.13		0.03	0.20		0.03	0.27		0.03	0.33		0.02
Min/Max/Σc_s(fF)	4000	**14**	**14**	**12**	**10**	**16.7%**	**11**	**9**	**18.2%**	**11**	**9**	**18.2%**	**11**	**9**	**18.2%**	**9**
		0.03	0.06	0.03	0.13		0.04	0.20		0.03	0.27		0.03	0.31		0.02
7.97/7.97/15494	8000	**6**	**6**	**4**	**4**	**0.0%**	**4**	**4**	**0.0%**	**4**	**4**	**0.0%**	**4**	**4**	**0.0%**	**4**
		0.03	0.06	0.04	0.12		0.03	0.20		0.03	0.25		0.04	0.32		0.03
#Sinks=2400	500	**133**	**133**	**121**	**105**	**13.2%**	**115**	**102**	**11.3%**	**116**	**100**	**13.8%**	**116**	**99**	**14.7%**	**99**
		0.04	0.10	0.03	0.18		0.03	0.29		0.04	0.39		0.03	0.49		0.03
WL=97mm	1000	**62**	**62**	**54**	**50**	**7.4%**	**52**	**48**	**7.7%**	**52**	**47**	**9.6%**	**52**	**47**	**9.6%**	**47**
		0.03	0.07	0.03	0.16		0.04	0.27		0.04	0.37		0.04	0.45		0.02
C_b=37.50fF	2000	**29**	**29**	**26**	**24**	**7.7%**	**26**	**23**	**11.5%**	**25**	**23**	**8.0%**	**25**	**23**	**8.0%**	**23**
		0.03	0.08	0.03	0.16		0.04	0.25		0.04	0.35		0.03	0.44		0.02
Min/Max/Σc_s(fF)	4000	**16**	**16**	**15**	**12**	**20.0%**	**15**	**10**	**33.3%**	**14**	**10**	**28.6%**	**14**	**10**	**28.6%**	**10**
		0.03	0.07	0.04	0.16		0.04	0.25		0.04	0.33		0.04	0.42		0.03
7.97/7.97/19423	8000	**9**	**9**	**8**	**5**	**37.5%**	**7**	**5**	**28.6%**	**7**	**5**	**28.6%**	**7**	**5**	**28.6%**	**5**
		0.04	0.08	0.05	0.17		0.04	0.25		0.05	0.34		0.04	0.41		0.03
#Sinks=2600	500	**266**	**266**	**238**	**211**	**11.3%**	**229**	**204**	**10.9%**	**226**	**198**	**12.4%**	**227**	**196**	**13.7%**	**196**
		0.04	0.14	0.04	0.36		0.04	0.66		0.04	0.93		0.03	1.12		0.02
WL=150mm	1000	**125**	**125**	**117**	**104**	**11.1%**	**109**	**99**	**9.2%**	**106**	**98**	**7.5%**	**106**	**98**	**7.5%**	**97**
		0.03	0.12	0.04	0.32		0.04	0.55		0.04	0.79		0.04	0.98		0.03
C_b=37.50fF	2000	**64**	**64**	**55**	**50**	**9.1%**	**52**	**49**	**5.8%**	**52**	**48**	**7.7%**	**52**	**48**	**7.7%**	**48**
		0.04	0.12	0.04	0.31		0.05	0.55		0.04	0.78		0.04	0.97		0.02
Min/Max/Σc_s(fF)	4000	**34**	**34**	**30**	**26**	**13.3%**	**29**	**23**	**20.7%**	**28**	**22**	**21.4%**	**28**	**22**	**21.4%**	**22**
		0.04	0.12	0.04	0.32		0.05	0.55		0.04	0.78		0.05	0.97		0.03
2.96/63.57/45077	8000	**15**	**15**	**15**	**12**	**20.0%**	**13**	**11**	**15.4%**	**13**	**11**	**15.4%**	**13**	**11**	**15.4%**	**11**
		0.04	0.12	0.05	0.31		0.05	0.52		0.05	0.73		0.05	0.90		0.04
#Sinks=12000	500	**489**	**489**	**441**	**399**	**9.5%**	**424**	**375**	**11.6%**	**426**	**369**	**13.4%**	**423**	**366**	**13.5%**	**366**
		0.18	0.42	0.19	0.75		0.20	1.20		0.19	1.62		0.19	1.95		0.13
WL=452mm	1000	**227**	**227**	**208**	**185**	**11.1%**	**202**	**173**	**14.4%**	**202**	**171**	**15.3%**	**200**	**170**	**15.0%**	**170**
		0.19	0.34	0.21	0.69		0.21	1.12		0.21	1.52		0.21	1.86		0.15
C_b=37.50fF	2000	**114**	**114**	**100**	**89**	**11.0%**	**98**	**87**	**11.2%**	**98**	**86**	**12.2%**	**97**	**85**	**12.4%**	**85**
		0.19	0.36	0.22	0.70		0.24	1.11		0.22	1.46		0.23	1.80		0.16
Min/Max/Σc_s(fF)	4000	**53**	**53**	**48**	**45**	**6.2%**	**49**	**44**	**10.2%**	**49**	**44**	**10.2%**	**49**	**44**	**10.2%**	**44**
		0.21	0.35	0.22	0.67		0.25	1.02		0.24	1.37		0.26	1.70		0.18
4.55/4.55/54837	8000	**28**	**28**	**25**	**21**	**16.0%**	**25**	**20**	**20.0%**	**24**	**19**	**20.8%**	**24**	**19**	**20.8%**	**19**
		0.22	0.36	0.25	0.68		0.26	1.06		0.26	1.41		0.25	1.72		0.20

when combining X' with Y' and the proof is complete. Otherwise, we may assume that the updated number j of redundant buffers satisfies

$$s(X') \geq s(Y') + j. \qquad (4)$$

Similarly, if $l(X') < l(Y') + j$, then Z' can be replaced by the buffering Z'', obtained by moving $k = \min\{j, l(Y') + j - l(X')\}$ redundant buffers from the top of $br(w)$ to the top of $br(u)$. Z'' dominates Z because $nb(Z'') = nb(Z), l(Z'') = l(Z), \text{cap}(Z'') = C_b \leq$

$\mathrm{cap}(Z)$, and $s(Z'') = s(Z)$ by (4) (see Fig. 8). Again, if $k = j$, then Z'' is added by DP2 to $\mathcal{L}(u)$ when combining X' with Y', and the proof is complete. Otherwise, we may assume that

$$l(X') \geq l(Y') + j. \tag{5}$$

We now show that inequalities (4) and (5) imply that Z' is generated by Step 3(a) of DP2 when combining $X' \in \mathcal{L}(v)$ with $Y' \in \mathcal{L}(w)$. First, note that $j \geq t = \max\{0, l(X) - s(Y) - \Delta\}$ since Z' is feasible and, thus, $l(X') - s(Y') - j \leq \Delta$. Finally, Step 3a(*) of DP2 inserts j buffers at the top of $br(w)$, since, by (4) and (5), the intervals $[s(X'), l(X')]$ and $[s(Y') + j, l(Y') + j]$ are not strictly containing one another.

Finally, the theorem follows from the fact that only dominated or redundant bufferings are deleted in Steps 3(b) and (c) of DP2. Indeed, if Z' is deleted, then there exists a buffering $W \in \mathcal{L}(u)$ and $k \geq 2$, such that $nb(Z') = nb(W) + k, l(Z') = l(W) + k$, and $s(Z') = s(W) + k$. Since $nb(Z') \geq 2$, it follows that $\mathrm{cap}(Z') \geq C_b$, and thus, Z' is dominated by W with k buffers added at the top of $br(u)$. ■

Lemma 1: For each node u of T, the set $\mathcal{L}(u)$ computed by DP2 contains at most $2(\Delta + 1)\mathrm{NB}$ bufferings.

Proof: Let us call a triple (nb, l, s) of integers *represented* in $\mathcal{L}(u)$ if there exists a buffering $X \in \mathcal{L}(u)$ such that $nb(X) = nb$, $l(X) = l$, and $s(X) = s$. Since dominated bufferings are removed in Step 3(b), any triple of parameters (nb, l, s) can be represented at most once by the bufferings surviving in $\mathcal{L}(u)$ (by a buffering with the smallest possible residual capacitance). We will show that no more than $2(\Delta + 1)\mathrm{NB}$ triples (nb, l, s) can be represented. Indeed, consider all triples (nb, l, s) with $l - s = \delta$ and $nb - l = m$, i.e., triples of the form $(nb, nb - m, nb - m - \delta)$. For every fixed δ and m, there are *at most* two values of nb for which $(nb, nb - m, nb - m - \delta)$ will survive the deletions in Steps 3(c) of DP2. The lemma follows since all bufferings generated by the algorithm have $0 \leq \delta \leq \Delta$ and $0 \leq m < \mathrm{NB}$. ■

Theorem 2: DP2 returns the optimum buffering in time $O(n(\Delta + 1)^3\mathrm{NB}^2)$.

Proof: The running time follows by observing that, for each of the $n - 1$ nonsink nodes, DP2 needs $O((\Delta + 1)^3\mathrm{NB}^2)$ time to compute the set $\mathcal{L}(u)$. Indeed, the time needed by Step 3(a) is $O((\Delta+1) \cdot |\mathcal{L}(v)| \cdot |\mathcal{L}(w)|)$, where v and w are the two children of u. Lemma 1 implies that, at the end of Step 3(a), the size of $\mathcal{L}(u)$ is $M = 4(\Delta + 1)^3\mathrm{NB}^2$. To complete the proof, we need to show that Steps 3(b) and (c) can be implemented in $O(M)$ time. This is done as follows.

1) For each buffering X, compute $m(X) = nb(X) - l(X)$ and distribute Xs into NB buckets, each containing bufferings with the same m;
2) Distribute all bufferings in each bucket between $\Delta + 1$ subbuckets, each containing bufferings with the same skew $\delta \in \{0, 1, \ldots, \Delta\}$.
3) In one linear traversal, extract from each subbucket two bufferings: a buffering with a minimum number of buffers nb and, subject to this, a minimum residual capacitance, plus, if it exists, a buffering with $nb + 1$ buffers and a residual capacitance equal to C_b (all other bufferings are either dominated or redundant). ■

V. Experimental Results

Both DP2 and the greedy algorithm of [10] have been implemented in C. Table I gives the results obtained by running the two algorithms on six testcases from [2]. In this set of experiments, the initial tree was computed using the Greedy-DME algorithm of [3] with linear delay. The unit-wire capacitance was $C_w = 0.177\ fF/\mu$m and the buffer input capacitance was $C_b = 37.5\ fF$. The first column of Table I gives the total wirelength of the Greedy-DME tree (WL) and the minimum, maximum, and total sink input capacitance for each instance (sink capacitances vary between 2.04 fF and 63.57 fF in these testcases). Reported runtimes are for a SUN Ultra 60 running SunOS 5.7.

TABLE II
Min/Max SPICE Insertion Delay and Skew (in Picoseconds) for Greedy-DME (Linear Delay) Unbuffered Trees and Their Optimum Bufferings With $\Delta \in \{0, 1, 2, 3\}$

#Sinks	C_U(fF)	Delay	$\Delta = 0$	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	Unbuffered
		Max	390	442	490	490	1943
330	500	Min	277	290	274	274	1590
		Skew	**113**	**152**	**216**	**216**	**353**
		Max	483	494	644	644	1943
330	1000	Min	336	273	316	316	1590
		Skew	**147**	**220**	**327**	**327**	**353**
		Max	758	754	847	873	6706
830	500	Min	648	599	589	575	4153
		Skew	**110**	**154**	**258**	**298**	**2553**
		Max	773	841	911	989	6706
830	1000	Min	618	598	601	589	4153
		Skew	**154**	**242**	**309**	**400**	**2553**

TABLE III
Min/Max SPICE Insertion Delay and Skew (in Picoseconds) for Greedy-DME (Elmore Delay) Unbuffered Trees and Their Optimum Bufferings With $\Delta \in \{0, 1, 2, 3\}$

#Sinks	C_U(fF)	Delay	$\Delta = 0$	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	Unbuffered
		Max	377	408	433	495	1860
330	500	Min	308	275	234	228	1848
		Skew	**69**	**132**	**198**	**266**	**12**
		Max	422	465	565	583	1860
330	1000	Min	307	299	282	226	1848
		Skew	**115**	**165**	**282**	**356**	**12**
		Max	785	806	843	919	6627
830	500	Min	694	661	657	666	6567
		Skew	**90**	**145**	**186**	**252**	**59**
		Max	825	884	973	973	6627
830	1000	Min	657	689	723	723	6567
		Skew	**167**	**195**	**249**	**249**	**59**

The first observation is that, although slower than the greedy algorithm of [10] by a factor of up to 20 ×, DP2 has very practical runtime (even for the 12 000-sink testcase, DP2 finishes in less than 2 s). The results suggest that the worst case bound in Theorem 2 is an overly pessimistic estimation of actual runtime. Indeed, in our experiments, the average size of $\mathcal{L}(u)$'s was always significantly smaller than the bound given in Lemma 1.

As expected, both algorithms insert the optimum number of buffers when a buffer skew bound of zero is imposed. For nonzero skew bounds, DP2 inserts almost always strictly fewer buffers compared to the greedy algorithm of [10], with savings reaching as much as 37.5%. Table I also shows that a significant reduction in the number of inserted buffers can be achieved with a small increase in buffer skew, e.g., when going from zero buffer skew to a buffer skew of one. For comparison, we have also included in the table a lower bound on the number of buffers, which is the minimum number of buffers needed to meet the load-cap constraints while disregarding buffer skew constraints. This lower bound was computed using the linear-time algorithm given in [2]. In all but one case, the lower bound is matched by the optimum buffering with $\Delta = 4$, and often it is matched with a buffer skew as small as two.

The effectiveness of the buffer-skew model was verified by SPICE simulation based on the 130-nm ITRS Predictive Technology Beta Version device model. In these simulations, buffers were formed as pairs of inverters. Interconnect wire segments were represented by a Π-model with 0.076 Ω unit-wire resistance and equal wire capacitance lumped at both ends of the segment. Each interconnect was driven by a ramped input signal with 150-ps slew time under 1.5 V supply voltage. Tables II and III show the maximum, minimum, and skew of 50% SPICE insertion delay from the source to each sink for trees constructed using the Greedy-DME algorithm with linear, respectively, Elmore delay. As expected, the more accurate Elmore delay leads to much smaller skew values. Delay skews after buffering are relatively small, and can be further reduced by optimizations that do not affect the number of buffers, e.g., fine tuning of load buffers. Furthermore, the results on both types of trees exhibit a strong correlation between buffer skew and delay skew, thus justifying the use of the buffer skew model for early estimation of buffering resources.

VI. Conclusion and Future Research

In this paper, we have addressed the problem of finding the minimum-buffered routing of a given tree under buffer load and skew constraints. We have shown that a greedy algorithm previously proposed for this problem in [10] may fail to find the optimum solution, and we have proposed an exact dynamic programming algorithm. Experimental results on test cases extracted from recent industrial designs show that the dynamic programming algorithm has practical running time and inserts up to 37.5% fewer buffers compared to the greedy algorithm of [10].

Our future research will address:

1) multiconstraint formulations in which, e.g., input capacitance and fanout must be upper-bounded simultaneously;
2) minimum inverter insertion in a given tree subject to sink-polarity constraints, in addition to inverter load and skew constraints;
3) simultaneous tree construction and buffering under given buffer load and skew constraints.

References

[1] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 588–593.
[2] C. Alpert, A. B. Kahng, B. Liu, I. I. Măndoiu, and A. Z. Zelikovsky, "Minimum-buffered routing for slew and reliability control," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 2001, pp. 408–415.
[3] M. Edahiro, "Delay minimization for zero-skew routing," in *Proc. IEEE-ACM Int. Conf. Computer-Aided Design*, 1993, pp. 563–567.
[4] P. Fang, J. Tao, J. F. Chen, and C. Hu, "Design in hot-carrier reliability for high performance logic applications," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1998, pp. 525–532.
[5] C. Hu, "Hot carrier effects," in *Advanced MOS Device Physics*, N. G. Einspruch, Ed. New York: Academic, 1989, pp. 119–160.
[6] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma, "Interconnect tuning strategies for high-performance ICs," in *Proc. Conf. Design Automation Test Eur.*, Feb. 1998, pp. 471–478.
[7] J. Lillis, C.-K. Cheng, and T.-T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, pp. 437–447, 1996.
[8] S. Rzepka, K. Banerjee, and E. Meusel, "Characterization of self-heating in advanced VLSI interconnect lines based on thermal finite element simulation," *IEEE Trans. Comp., Packag., Manufact. Technol. A*, vol. 21, pp. 406–411, Sept. 1998.
[9] L. Scheffer, private communication, Apr. 2000.
[10] G. E. Tellez and M. Sarrafzadeh, "Minimal buffer insertion in clock trees with skew and slew rate constraints," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 333–342, Apr. 1997.
[11] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.
[12] A. Z. Zelikovsky and I. I. Măndoiu, "Practical approximation algorithms for zero- and bounded-skew trees," *SIAM J. Discrete Math.*, vol. 15, no. 1, pp. 97–111, 2002.

A New Reasoning Scheme for Efficient Redundancy Addition and Removal

Chih-Wei Jim Chang, Ming-Fu Hsiao, and Malgorzata Marek-Sadowska

***Abstract*—Redundancy addition and removal is a rewiring technique which, for a given target wire w_t, finds a redundant alternative wire w_a. The addition of w_a makes w_t redundant and, hence, removable without changing the overall circuit functionality. Incremental logic restructuring based on this technique has been used in many applications. However, in the earlier methods, the search for valid alternative wires required trial-and-error redundancy testing of a potentially large set of candidate wires. Here, we study the fundamental theory behind this technique and propose a new reasoning scheme (RAMFIRE), which directly identifies alternative wires without performing trial-and-error tests. Experimental results show speedup of up to 15 times than that of the best techniques in the literature.**

***Index Terms*—Logic restructuring, logic synthesis, physical synthesis, timing optimization.**

I. Introduction

Redundancy addition and removal (RAR) is a powerful combinational logic restructuring technique [7]. First, a redundant wire is added to the circuit. As a result, some previously irredundant wires become redundant and, hence, can be removed without affecting the overall functionality of the circuit. The underlying engine is based on logic implication. Many applications of this technique have been developed in the past, including technology-independent literal minimization [2], [3], [7], [13], field programmable gate array routing [5], and postlayout timing optimization [9], [11]. The major advantage of the RAR technique is that only wires are reconnected while logic gates are preserved. This property is especially desirable in the deep-submicron age, when timing estimation obtained during logic synthesis cannot be justified after placement and routing. Timing can be incrementally corrected through a sequence of rewiring steps guided by accurate physical information. Rewiring minimally perturbs layout and helps in achieving timing closure.

Manuscript received March 6, 2002; revised November 1, 2002. This work was supported in part by the Semiconductor Research Corporation under Grant 98-DJ-619. This paper was recommended by Associate Editor E. Macii.

C.-W. J. Chang was with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA. He is now with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: emersons@ms6.hinet.net).
M.-F. Hsiao was with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA. He is now with Cadence Design Systems, San Jose, CA 95134 USA (e-mail: cwchang1@yahoo.com).
M. Marek-Sadowska is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA.

Digital Object Identifier 10.1109/TCAD.2003.814239